SUPERVISED LEARNING ON RELATIONAL DATABASES WITH GRAPH NEURAL NETWORKS

Milan Cvitkovic
Department of Computing and Mathematical Sciences
California Institute of Technology
mcvitkov@caltech.edu

ABSTRACT

The majority of data and machine learning scientists and engineers use relational data in their work (State of ML and Data Science 2017, Kaggle, Inc.). But training machine learning models on data stored in relational databases requires significant data extraction and feature engineering efforts. These efforts are are not only costly, but they also destroy potentially important relational structure in the data. We introduce a technique based on Graph Neural Networks that circumvents these efforts and disadvantages. Without any feature engineering, hyperparameter optimization, or model selection, our model performs adequately compared to expert practitioners on two Kaggle competitions.

1 INTRODUCTION

Relational data is the most widely used type of data across all industries (Kaggle, Inc., 2017). Second only to web technologies,[1] relational databases (RDBs) are the most popular technology among developers (Stack Exchange, Inc., 2018). The market merely for hosting RDBs was around $45 billion USD in 2017 (Asay, 2016), which is to say nothing of the societal and economic value of the data contained in those RDBs.

Yet learning on data in RDBs has not been a focus of deep learning in recent years. The standard approach toward RDBs in machine learning is to "flatten" the relational data they contain into tabular form, since standard supervised learning models expect their inputs to be fixed–size vectors. This flattening process not only requires significant preprocessing and feature–engineering effort, but it destroys relational information present in the data. Recent developments in deep learning for relational data present an opportunity for alleviating these issues.

In this work we present a model that uses Graph Neural Networks (GNNs) to perform supervised learning directly on data in relational databases, with no need for manual data extraction or flattening. We find that our model performs adequately compared to expert practitioners on two Kaggle competitions.

2 BACKGROUND

2.1 RELATIONAL DATABASES

An example RDB[2] is shown in the top part of Figure 1.

An RDB is a collection of related tables. Each table is composed of a number of rows. All the rows in a given table have the same format; each one is a single entry in the table. The format of the rows in a table is specified by that table's columns. Each column contains one type of data, like strings, ints, floats, or dates.

[1] HTML, CSS, and Javascript

[2] RDBs are sometimes informally called "SQL databases". SQL (Structured Query Language) is the language most commonly used to get data into and out of RDBs.

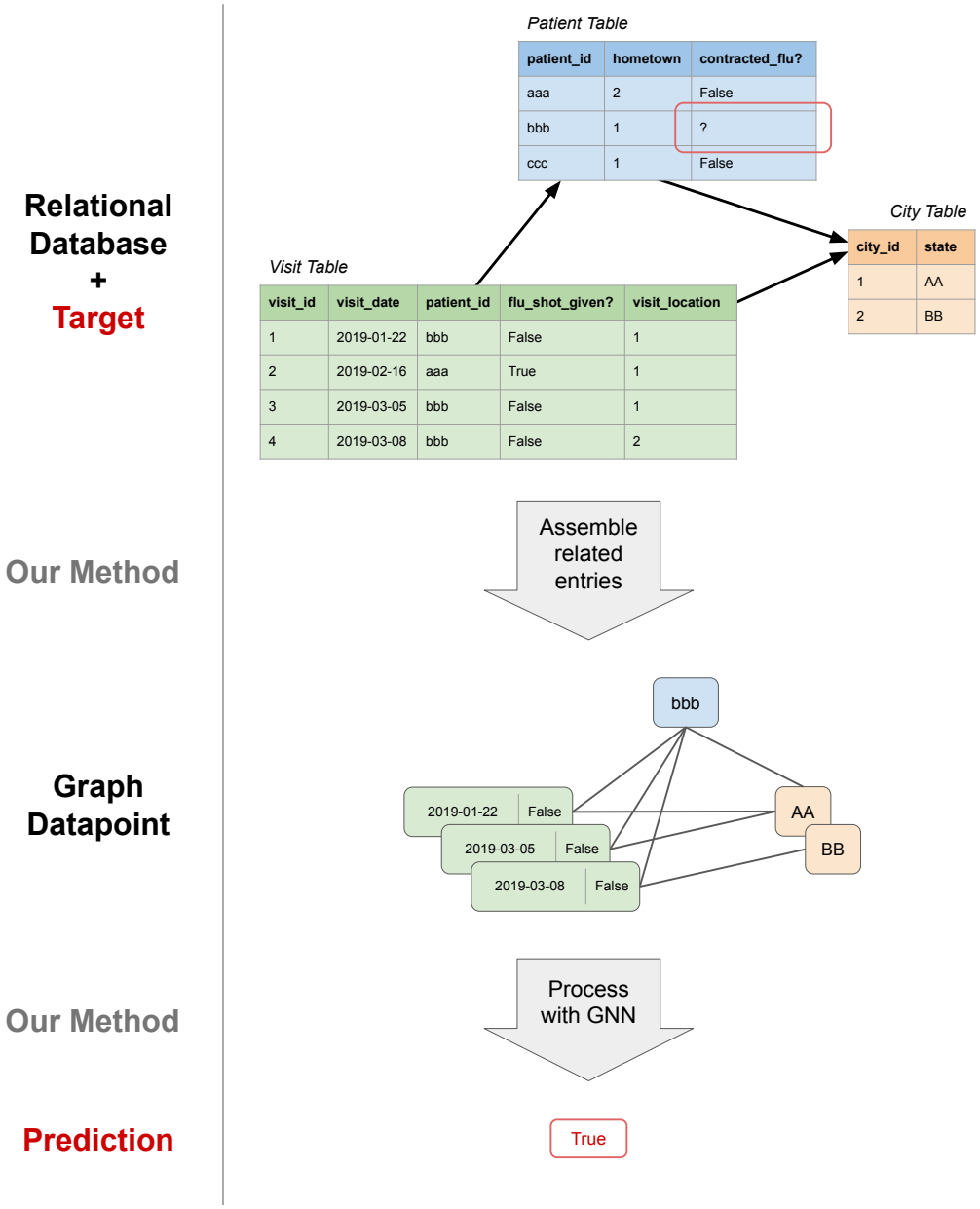

Figure 1: Example of our method making a prediction on a miniature RDB containing medical records. Given an RDB and a target to be predicted in it (in this example, whether patient "bbb" got the flu), our method first assembles all related information in the RDB in the form of a graph. This graph is processed by a Graph Neural Network to produce a prediction.

What makes RDBs "relational" is that a column in one table can refer to a column in another table. For example, in Figure 1 the `patient_id` column in the Visit Table refers to the `patient_id` column in the Patient Table. A column like this that refers to another table is called a *foreign key*. The value of the `patient_id` foreign key in a particular row in the Visit Table indicates which patient came for that visit.

Readers comfortable with object–oriented programming should think of each table as an object class. The table's columns are the class's attributes, and each of the table's rows is an instance of that class. A foreign key is an attribute that refers to an instance of a different class. This is an entirely equivalent way of thinking about RDBs.

There are many software systems for creating and managing RDBs, including MySQL, PostgreSQL, and SQLite. But effectively all RDB systems adhere closely (though not perfectly) to the same technical standard (International Organization for Standardization, 2016) that defines how they are structured and what data types they can contain. Thanks to this nonproliferation of standards, our model is applicable to supervised learning problems in nearly every RDB in use today.

2.2 Graph Neural Networks

A Graph Neural Network (GNN) is any deep, differentiable, parameterized model that takes graphs as input. Many types of GNN have been introduced in the literature, and several nomenclatures and taxonomies have been proposed for organizing them. Useful surveys of GNNs can be found in Gilmer et al. (2017), Battaglia et al. (2018), and Wu et al. (2019). For brevity, we defer explanation of how GNNs operate to those references.

2.3 Supervised learning tasks on relational databases

In this paper, "supervised learning on an RDB" means training a model to predict the value in a particular column in a particular table in the RDB, given all other relevant information in the RDB.

For example, in the Valued Shoppers Kaggle Competition used in the Experiments below, one of the tables in the Valued Shoppers RDB has a column `repeater`, which contains a Boolean value indicating whether that shopper was a repeat customer. The value of `repeater` is provided for some customers (the training set), and the goal of the competition is to predict the value of `repeater` for a set of other customers (the test set).

In general, one might want to train a model to predict a more complicated function of the RDB's contents than simply the value of one column. And in practice when working with a real RDB one needs to be careful about target leakage. We ignore these details in this work, but the system we use to address this more general case is summarized in Supplementary Information Section 6.2.

3 Model

A depiction of our model making a prediction is shown in Figure 1. In brief: given an RDB and a row containing a value we want to predict, our model collects all information in the RDB that is relevant to the target, assembles it into a graph, and then processes this graph with a GNN to produce a prediction.

3.1 RDB to graph

Suppose our model is trying to predict the value of column j in the ith row in table T. The set of rows in the RDB that might contain relevant information for making this prediction is the set of rows for which there exists a path of foreign keys (treated as undirected edges) that leads to row i. Our model queries the database to find these rows and assembles them into a graph where each row is a vertex in the graph. The data stored in the row a are the features of vertex a. Vertex a is connected to vertex b if row a has foreign key referencing row b.

For example, Figure 1 shows our model predicting the value of `contracted_flu` for the 2nd row in the Patient Table. The graph containing all information relevant to making this prediction contains this 2nd row itself, along with all 3 visits from the Visit Table that this patient made, and the entries from the City Table referenced by the visit or patient rows. Note that the vertices only contain features that correspond to non–foreign–key columns; the foreign–key information is represented by the edges to other vertices.

3.2 Initializing the graph for the Graph Neural Network

Once the input graph has been assembled, our model needs to convert the features of each vertex into a real–valued vector in $\mathbb{R}^d$, where d is the dimension of the GNN's hidden states. To do this, for each vertex, our model (1) vectorizes the features for each of the vertex's columns according to

the column's data type, (2) concatenates these vectors, (3) puts them through a 1–layer MLP with output dimension $\mathbb{R}^d$.

To vectorize `float`– or `int`–type columns that represent scalar data, our model leaves them unchanged. To vectorize `integer` or `string` columns that represent categorical information, our model uses a trainable embedding matrix. And to vectorize `datetime` columns, our model converts them to a standard set of date features; see Supplementary Materials Section 6.1.

4 EXPERIMENTS

Code to reproduce all experiments can be found online.[3]

We assess our method by seeing how it performs on two Kaggle competitions. Both competitions — Inventory Demand[4] and Valued Shoppers[5] — are supervised learning tasks on data in relational databases. We compare our method to the performance of teams that participated in the competition.

We use a Graph Convolutional Network (GCN) (Kipf & Welling, 2016) with 5 layers as the GNN in our model. Our model produces a prediction from the output of the GCN by taking the mean of all vertices' hidden states and applying a linear mapping. The output of this mapping is considered to be a scalar prediction for the Inventory Demand challenge and to be logits for the Valued Shopper challenge. The GCN was trained by SGD using the Adam optimizer (Kingma & Ba, 2014) with a learning rate of $3\mathrm{e}{-4}$.

Results are shown in Table 1.

Table 1: Performance on two Kaggle competitions whose datasets are relational databases. The combined prize money for these competitions was $55,000, and the Kaggle Teams had over two months to work on their models. In contrast, our model's results, while unexceptional, were produced in under four days using a simple, off–the–shelf GNN and with no feature engineering, hyperparameter tuning, or model selection.

		Kaggle Teams			
Competition	Metric	Simple Prior	Median	Best	GNN (ours)
Inventory Demand	RMSLE ↓	0.963	0.517	0.443	0.636
Valued Shopper	AUROC ↑	0.520	0.583	0.627	0.577

5 RELATED WORK

The only work we are aware of that studies supervised learning tasks on relational databases of the sort considered above comes from the feature engineering literature.

Kanter & Veeramachaneni (2015) present a system that automatically engineers features to help predict the value in a user–selected target column in a relational database. These engineered features can then be used as inputs to any standard machine learning algorithm. The engineered features consist of quantizations and embeddings of the other columns in the same table as the target column, plus additional features aggregated from other tables. These aggregated features are produced by recursively applying predefined aggregation functions like `MAX` or `SUM` to rows connected to the target column by foreign key relationships. This is somewhat analogous to the multi–view learning approach of Guo & Viktor (2008).

Lam et al. (2017) and Lam et al. (2018) both extend Kanter & Veeramachaneni (2015), the former by expanding the types of feature quantizations and embeddings used, and the latter by using Recurrent Neural Networks as the aggregation functions rather than functions like `MAX` and `SUM`.

[3] `https://github.com/mwcvitkovic/Supervised-Learning-on-Relational-Databases-with-Graph-Neural-Networks`

[4] `https://www.kaggle.com/c/grupo-bimbo-inventory-demand`

[5] `https://www.kaggle.com/c/acquire-valued-shoppers-challenge`

Other thematically–related areas include Statistical Relational Learning (Getoor & Taskar, 2007), which is concerned with learning probabilistic models of structured data, and some works in the Probabilistic Programming literature that study how to compile RDBs into probabilistic graphical models (Singh & Graepel, 2012; Gordon et al., 2014; Mansinghka et al., 2015). The probabilistic models studied in these works could in principle be used for supervised learning tasks of the sort we consider in this work.

REFERENCES

Matt Asay. NoSQL keeps rising, but relational databases still dominate big data, April 2016. URL https://www.techrepublic.com/article/nosql-keeps-rising-but-relational-databases-still-dominate-big-data/.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261 [cs, stat]*, June 2018. URL http://arxiv.org/abs/1806.01261. arXiv: 1806.01261.

Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning*. MIT Press, 2007. ISBN 978-0-262-07288-5.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1263–1272. JMLR. org, 2017.

Andrew D Gordon, Thore Graepel, Nicolas Rolland, Claudio Russo, Johannes Borgstrom, and John Guiver. Tabular: a schema-driven probabilistic programming language. In *ACM SIGPLAN Notices*, volume 49, pp. 321–334. ACM, 2014.

Hongyu Guo and Herna L Viktor. Multirelational classification: a multiple view approach. *Knowledge and Information Systems*, 17(3):287–312, 2008.

International Organization for Standardization. ISO/IEC 9075-1:2016: Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework), December 2016. URL http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/35/63555.html.

Kaggle, Inc. The State of ML and Data Science 2017, 2017. URL https://www.kaggle.com/surveys/2017.

J. M. Kanter and K. Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 1–10, October 2015. doi: 10.1109/DSAA.2015.7344858.

Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. URL http://arxiv.org/abs/1412.6980. arXiv: 1412.6980.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *ICLR 2017*, September 2016. URL http://arxiv.org/abs/1609.02907. arXiv: 1609.02907.

Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. One button machine for automating feature engineering in relational databases. *arXiv:1706.00327 [cs]*, June 2017. URL http://arxiv.org/abs/1706.00327. arXiv: 1706.00327.

Hoang Thanh Lam, Tran Ngoc Minh, Mathieu Sinn, Beat Buesser, and Martin Wistuba. Neural Feature Learning From Relational Database. *arXiv:1801.05372 [cs]*, January 2018. URL http://arxiv.org/abs/1801.05372. arXiv: 1801.05372.

Vikash Mansinghka, Richard Tibbetts, Jay Baxter, Pat Shafto, and Baxter Eaves. BayesDB: A probabilistic programming system for querying the probable implications of data. *arXiv:1512.05006 [cs]*, December 2015. URL http://arxiv.org/abs/1512.05006. arXiv: 1512.05006.

Sameer Singh and Thore Graepel. Compiling Relational Database Schemata into Probabilistic Graphical Models. *arXiv:1212.0967 [cs, stat]*, December 2012. URL http://arxiv.org/abs/1212.0967. arXiv: 1212.0967.

Stack Exchange, Inc. Stack Overflow Developer Survey 2018, 2018. URL https://insights.stackoverflow.com/survey/2018/.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *arXiv:1901.00596 [cs, stat]*, January 2019. URL http://arxiv.org/abs/1901.00596. arXiv: 1901.00596.

6 SUPPLEMENTARY MATERIAL

6.1 EMBEDDING DATES AND TIMES

Our model converts a datetime to a vector by concatenating the following commonly used date and time features:

1. Year (scalar value)
2. Month (one–hot encoded)
3. Week (one–hot encoded)
4. Day (one–hot encoded)
5. Day of week (one–hot encoded)
6. Day of year (scalar value)
7. Month end? (bool, one–hot encoded)
8. Month start? (bool, one–hot encoded)
9. Quarter end? (bool, one–hot encoded)
10. Quarter start? (bool, one–hot encoded)
11. Year end? (bool, one–hot encoded)
12. Year start? (bool, one–hot encoded)
13. Day of week $\cos$ (scalar value)
14. Day of week $\sin$ (scalar value)
15. Day of month $\cos$ (scalar value)
16. Day of month $\sin$ (scalar value)
17. Month of year $\cos$ (scalar value)
18. Month year $\sin$ (scalar value)
19. Day of year $\cos$ (scalar value)
20. Day of year $\sin$ (scalar value)

The $\cos$ and $\sin$ values are for representing cyclic information, and are given by computing $\cos$ or $\sin$ of $2\pi\frac{value}{period}$. E.g. "day of week $\cos$" for Wednesday, the third day of seven in the week, is $\cos(2\pi\frac{3}{7})$.

6.2 SYSTEM USED FOR SPECIFYING TASK AND TRAINING SETUP GIVEN AN RDB

We define a `PREDICT` statement that can be used to specify a supervised learning task on an RDB.

A `PREDICT` statement is just like a SQL `SELECT` statement but with an additional `GIVEN...USING ONLY` clause that delimits what information the model will receive as input during training and testing.

The `PREDICT` statement specifies everything needed to construct and train the model. Specifically:

1. The `GIVEN` values, limited by their `WHERE` clause, specify which column values the SGD mini batches will be sampled from during training.
2. The `PREDICT` statement (with `GIVEN` values filled in) specifies what SQL `SELECT` query to execute to get the ground truth for the training loss function for a particular set of `GIVEN` values. The `PREDICT` statement has identical syntax to a (dynamic) SQL `SELECT` statement: it can include `WHERE` clauses, variables, `JOIN` clauses, conditionals like `AND`, functions like `SUM` or `COUNT`, etc.
3. The `USING ONLY` statement specifies what SQL `SELECT` query (or queries) to execute in order to get the input data for the model. One could also use an `EXCLUDING` clause if it is easier to tell the model what not to include, and it will automatically include everything else.

Example:

Say we wanted to look up how many toothbrushes Alice sold the week after January 5th. We'd execute the SQL `SELECT` statement

```
SELECT COUNT(sale_id)
FROM sales_table
WHERE prod_id = toothbrush,
    salesperson_id = Alice,
    (sale_date >= '2019-01-05' AND sale_date <= ('2019-01-05' + 7))
```

to find out.

Now say instead we want to train a model that will *predict* how many toothbrushes *any* salesperson will sell a week after *any* date, using only their last 30 days of toothbrush sale data to do so. We simply modify the above `SELECT` statement into this `PREDICT` statement:

```
PREDICT COUNT(sale_id)
FROM sales_table
WHERE prod_id = toothbrush,
    salesperson_id = @salesperson,
    (sale_date >= @date AND sale_date <= (@date + 7))
GIVEN @salesperson NUMBER(6),
    @date DATE
WHERE @date < GETDATE() - 7
USING ONLY prod_id = toothbrush,
    salesperson_id = @salesperson
    (sale_date <= @date AND sale_date >= (@date - 30))
```

Example: Predict number of days a customer is overdue on their payments after their first month given their first two week's behavior

```
PREDICT COUNT(customer_daily_benchmark_id)
FROM customer_daily_benchmark_table
WHERE overdue_days > 0,
    days_elapsed <= 30,
    customer_id = @customer
GIVEN @customer NUMBER(6)
USING ONLY customer_id = @customer,
    days_elapsed <= 14
```

Example: Predict number of toothbrushes that will be sold from a particular warehouse in a given month

```
PREDICT COUNT(sale_id)
FROM sales_table
WHERE prod_id = toothbrush,
    warehouse_id = @warehouse,
    MONTH(sale_date) = @month
GIVEN @warehouse NUMBER(6),
```

```
    @month DATE
WHERE @month < MONTH(GETDATE())
USING ONLY warehouse_id = @warehouse,
    MONTH(sale_date) < @month
```

Example: Predict which product a salesperson will sell most of

```
PREDICT max(sale_id)
FROM sales_table
WHERE prod_id = toothbrush,
    salesperson_id = @salesperson,
    (sale_date >= @date AND sale_date <= (@date + 7))
GIVEN @salesperson NUMBER(6),
    @date DATE
WHERE @date < GETDATE() - 7
USING ONLY prod_id = toothbrush,
    salesperson_id = @salesperson
    (sale_date <= @date AND sale_date >= (@date - 30)),
```

Example: Predict what disease a patient has based on everything we know about them

```
PREDICT diagnosis_id
FROM visit_records
INNER JOIN patient_data ON visit_records.patient_id = patient_data.id
WHERE attended_visit = True,
    diagnosis_made = True,
    patient_data.research_consent = True,
    visit_id = @visit
GIVEN @visit VARCHAR(16)
EXCLUDING diagnosis_id
```